Naval Oceanographic and Atmospheric Research Laboratory
NOARL Report 1 April 1990

(2)

Design of a Distributed Microprocessor Sensor System

B. S. Bourgeois
M. M. Harris
P. B. Wischow
J. H. Ross
Mapping, Charting, and Geodesy Division
Ocean Science Directorate

Foreword

The Geophysical Airborne Survey System (GASS) is a distributed microprocessor system that integrates magnetic, hydrographic, altitude, position, and attitude sensors. GASS is used on the Project Magnet P-3 Orion aircraft operated by the Naval Oceanographic Office to collect magnetic and hydrographic data. The magnetic data will be used to construct isomagnetic charts; the gravity data will be used to improve the accuracy of inertial navigation charts. Magnetic and water depth data will be used to upgrade navigational charts.

This report summarizes the research done in the area of distributed microprocessor systems. This study provides a foundation for the design of GASS and other systems of this type. Details of the final GASS design are provided.

W. B. Moseley
Technical Director

J. B. Tupaz, Captain, USN
Commanding Officer

Executive Summary

The Geophysical Airborne Survey System (GASS), developed by the Naval Ocean Research and Development Activity*, is a real-time, distributed microprocessor sensor system. The Naval Oceanographic Office intends to use this system on the Project Magnet P-3 Orion aircraft to collect worldwide magnetic and hydrographic data.

This report briefly discusses the mission and the history of GASS and reviews the technology advances of the last decade in the area of real-time distributed microprocessor systems. Specific topics include the goals of distributed system design, the qualitative value of distributed system design, and reliable design techniques for the hardware and software in distributed systems. This technology review provides a foundation for the GASS design.

The systematic design decisions made for GASS are detailed, and the system's architecture is described. Detailed drawings and descriptions of the hardware and software for the final GASS design are also included. Recommendations are made for possible system enhancements and for areas that require further investigation.

* Now the Naval Oceanographic and Atmospheric Research Laboratory (NOARL).

Acknowledgments

The authors acknowledge the Naval Oceanographic Office for funding this project under program element number 980101, managed by Mr. Roger Young. Special thanks are given to the software development team at Sverdrup Technology, Inc., under the supervision of Mr. Alan Guess, for their work on the real-time software. The authors also give special thanks to Mr. Jeffrey Spear of Planning Systems, Inc., for his work on the Operator Interface/System Control software. We thank Dr. Herbert C. Eppert, Jr., and Dr. Daniel G. Hickman of NOARL for their interest and support of the GASS project.

Contents

Design of a Distributed Microprocessor Sensor System

I. Introduction

This report discusses the design of the Geophysical Airborne Survey System (GASS). To this end, relevant research in the area of real-time distributed processing systems is reviewed, the major design considerations of GASS are discussed, and the final design is presented in detail.

A. The System and Its Mission

The GASS is a sensor system built around a distributed microprocessor network. This design was chosen to provide a high degree of system modularity. Modularity will enhance the system's capability to be flexible, available, repairable, and updated throughout its life span. GASS will be used around the world for surveying the earth's magnetic field and coastal water depths. The magnetic data will be used to construct isomagnetic charts, and the gravity data will be used to improve the accuracy of inertial navigation systems. Both the magnetic data and the water depth data will be used for navigational charts. In addition to the magnetic, gravity, and water depth sensors, GASS also includes position, altitude, attitude, and time devices needed to correlate the collected data.

B. System History

The GASS is designed for installation on the Project Magnet P-3 Orion aircraft, which is owned and operated by the Naval Oceanographic Office (NAVOCEANO). Project Magnet was initially used to conduct surveys of the earth's magnetic field[1]. This project was initiated on a trial basis by the Chief of Naval Operations in 1951, and made into a permanent program in 1957. The sensor system was upgraded in 1970 by the Applied Physics Laboratory of Johns Hopkins University, and was designated the *Geomagnetic* Airborne Survey System. The latest version of the system, the *Geophysical* Airborne Survey System, will be used to collect magnetic and hydrographic data. Hydrographic instruments are included in the system because of the 200-year backlog in coastal surveys (coastal surveys are presently conducted using boats). Gravity data will also be collected with this system when airborne gravimetric sensors become available. The new GASS, funded by NAVOCEANO, was developed by the former Naval Ocean Research and Development Activity (NORDA). (NOTE: NORDA has been designated as the Naval Oceanographic and Atmospheric Research Laboratory). The current system upgrade project began in December 1986, and the system design was based on the state of the technology as of December 1987. It is estimated that 60% of the development costs will be used for software development and testing, and only 40% will be used for the purchase of hardware. The system is designed for a life expectancy of 10 years.

C. Synopsis of Related Work

Distributed processing systems for real-time applications are becoming a reality with the advent of low-cost, high-powered microprocessors. The professed gains of a distributed system are many: lower hardware cost, higher reliability, increased flexibility, and ease of modernization and expansion due to the inherent modularity. Many distributed real-time systems provide proof-of-concept for this design methodology. Langley[2] provides the results of the work performed under Navy contracts to create a distributed missile guidance and control system. The traditional use of analog circuitry and single-processor designs resulted in systems that were difficult and expensive to upgrade. Langley's goal was to develop a modular architecture that would increase system flexibility and reduce the cost of software development. Wilcock[3] summarizes the concepts for the development of digital control systems for combat aircraft. This joint effort between the Royal Aircraft Establishment and the British Aerospace Corporation was to develop a distributed processing system that would reduce system weight, reduce pilot workload, improve maintainability, and improve survivability. Shin[4] discusses the Distributed Microprocessor Airborne Computing System (DMACS) that was developed at the Rensselaer Polytechnic Institute. The DMACS is designed to be a combined computer system for high-performance

military aircraft, responsible for the functions of weapons, navigation and control. Feo[5] outlines the evaluation requirements for Intelligent Redundant Actuation System (IRAS) designs. IRAS research is NASA-sponsored and should achieve reduced flight-control computer loading by shifting the tasks of failure isolation and configuration management to micro-processors at the actuator level. Shin[6] discusses the preliminary research of the Integrated Multi-Robot System (IMRS). IMRS is a distributed processing system designed for manufacturing systems. IMRS is expected to outperform contemporary centralized controllers on the basis of physical space, computer capabilities, throughput, flexibility, and fault-tolerance. Gluch[7] and Kieckhafer[8] discuss the Multicomputer Architecture for Fault-Tolerance (MAFT) developed by the Bendix Aerospace Technology Center. MAFT was designed for maximum reliability in real-time control systems. It consists of several nodes connected by a broadcast bus network. Each node uses two processors, one for executive functions and one for applications programs. Fura[9] discusses the Integrated Fault-Tolerant Avionics Systems Computer (IFTAS) developed by Boeing for the next generation of space transport vehicles. IFTAS is a distributed network of processing nodes interconnected by a high-speed serial bus. Each node will have from one to four processors depending upon the level of reliability required by its functions. One of the most recent and significant proponents for distributed processing is the Space Station Columbus' Data Management System (DMS)[10,11,12]. The DMS will be responsible for data communications, data processing, data administration, data storage, data retrieval, and data presentation throughout the space station. The contract for the DMS has been awarded to McDonnell-Douglas. Their proposed design will use a 100-million bit per second (Mbps), token ring, fiber optic network that will connect all of the processing nodes in a ring topology.

While much significant work has been done in the area of real-time distributed processing systems, there is still no clear-cut methodology for designing these systems. Mejzak[13] points out that the distributed system technology has not yet achieved a "rigid definition." Only a few attempts have been made to quantize the value of distributed design approaches.

Pedar[14] presents a study that attempts to find an optimal tradeoff among fault tolerance, computing capacity, and cost in a distributed processor system. Vielcanet[15] provides an empirical study of current ground and airborne microprocessor-based distributed systems that could be applied to space systems. Mangoubi[16] presents a method for evaluating the performance of real-time distributed systems. This paper was a joint effort between MIT and the Draper Laboratory, and it specifically addresses the utilization of resources and the response time delays for processing tasks. Lala[17] presents the network testbed developed by Draper Laboratory. This testbed will be used to experiment with various network concepts to develop more advanced network communication systems for future spacecraft. A recent, broad research effort for the cost effectiveness of various distributed design approaches has been undertaken by the U.S. Air Force with its Modular Avionics System Architecture (MASA) program[18]. Part of this study involves determining what level of modularity can best benefit the Air Force. Brock[18] points out that mandated use of common modules in aircraft systems could actually result in increased weight, size, and cost over an optimum point design.

Intrinsic to any discussion of distributed real-time processing systems are methods of achieving fault tolerance. The most common method for achieving fault tolerance in these systems is through redundancy of sensors, actuators, and processors. This redundancy often occurs naturally in distributed processing systems. Gelderloos[19] describes the redundancy management of the Shuttle Craft flight control system. This system uses quadruple redundant processors and data buses, with dual and triple redundant sensors and actuators. It is designed to continue safe operation after two redundant system failures. Watson[20] summarizes the work performed by General Dynamics in the development of a reliable avionics control system. Reliability was achieved here through the use of redundant sensors, redundant actuators, redundant computers, and advanced self-testing features within the computers. Sievers[21] discusses the development of a fault-tolerant computer for the U.S. Navy. He claims that the use of fault-tolerant architectures can achieve lifetime costs from 1/5 to 1/60 of the baseline costs. The architecture proposed involved the use of multiple single-board computers, which would run identical tasks and periodically check each other's results. McGlone[22] summarizes the results of the Air Force's Full Authority Fault Tolerant Electronics Engine Control (FAFTEEC) program. This program's goal was to develop the most cost-effective architecture for reliable digital control of a gas-turbine engine. The architecture chosen used dual sensors, actuators, and computers. Each computer would have dual central processing units (CPU) to determine the existence of a failure. Hartman[23] proposes architecture for advanced

fly-by-wire commercial aircraft systems. This work was performed by Honeywell under a contract from the Ames Research Center. The recommended architecture involved redundant sensors, actuators, data buses, and processors. The processors are to be distributed throughout the system, and fault tolerance would be implemented using either task reassignment or parallel running tasks with voting. Dzwonczy[24] presents a flight control system for the Entry Research Vehicle (ERV). The architecture for this system, developed by Draper Laboratory and Langley Research Center, involves the use of a central fault-tolerant processor that is connected to redundant sensors and effectors.

Fault tolerance through redundancy in distributed processing systems is typically implemented through the use of multiple identical software tasks running on separate processors with some form of voting. An example of this implementation is the Software Implemented Fault-Tolerance (SIFT) Computer[25]. This form of redundancy places more of a software burden on the system, and may require an inordinate amount of the computer system's throughput. In the case of SIFT, the executive functions of the computer utilize 80% of the system throughput[8]. Because of the disadvantages of software-intensive fault tolerance, the trend has been toward hardware-intensive, fault-tolerant schemes[15]. The goal of hardware-intensive fault tolerance is to provide redundancy that is "invisible" to the software designer. Montgomery[26] discusses a fault-tolerant microprocessor that uses three processing modules. Each module is made from two microprocessors, and upon a detected module failure the spare is switched in and the system is restarted using a previously stored system state vector. Evans[27] discusses the design of a fault-tolerant microcomputer used by the Metro Fire Board in Melbourne, Australia. This system uses three microprocessors, each with identical peripheral cards. Under fault-free operations two processors perform the same fault-critical tasks and the third is used for noncritical tasks. If a discrepancy between the fault-critical processors occurs, then the computer switches to a voting scheme using all three processors. Yaacob[28] describes a fault-tolerant microcomputer that meets the requirements of civil avionics reliability. This design uses three microprocessors wired in a triply modular redundant structure, and a fourth processor as a powered standby spare. Smith[29] describes the design theory of the WG-DCS machine developed at the Draper Laboratory. This machine uses three identical processors that run the same software. The output is obtained through a hardware voting scheme. Ichikawa[30] describes a fault-tolerant computer for use in satellite applications. This computer will use four CPUs, redundant read-only memory (ROM), random-access memory (RAM), clocks, ports, and data buses. Lala[17] describes a quadruple-redundant processor developed at the Draper Laboratory. This processor has been designed to maintain fault-free operation in the event of any single point failure. As discussed by Lala[17], there have been instances of failures in triply redundant flight control systems. Quadruple redundance is required to maintain fault-free operation in the presence of a single Byzantine (malicious) failure[31].

An abundance of research has been conducted on the development of software methods that will improve the reliability of real-time distributed systems. Two of these methods, multiversion software and recovery blocks, are intended to prevent system failures due to undetected residual programming errors. The multiversion software method involves writing several different versions of the same software task. If one version fails, then the other versions will maintain system operation. The recovery block method involves periodically saving the system's state vector and, in the event of a failure, restarting the system at its last saved state. A third method, specifically designed for distributed systems, is relocatable software. This software method enables a distributed processor system to tolerate hardware faults by relocating the tasks of the failed processors to operational processors. Multiversion software for real-time systems is discussed by Shepherd[32], Hitt[33], Avizienis[34], and Kieckhafer[8]. Recovery Block techniques for real-time systems are discussed by Anderson[35], Schneider[36], and Hitt[33]. Clarke[37], Schmid[38], Loques[39] and Best[40] discuss the use of relocatable software for real-time systems.

Despite the research that has been done on multiversion software and on recovery blocks, risks posed by these methods are still generally considered too high for use in distributed systems. Eckhardt[41] indicates that there are no data available to determine the cost effectiveness of multiversion software, even though this method is used in the space shuttle and in Canadian Nuclear Reactor systems. Voigt[42] states that exhaustive testing is the only effective method to date for generating reliable software. Avizienis[34] notes that multiversion software has been used in flight control systems for the Boeing 737/300, the Airbus, and the ATR aircraft. The authors of this paper reviewed the results of a University of California at Los Angeles (UCLA) and UC-Irvine study on multiversion software. The UCLA results were promising, but the UC-Irvine study cast serious doubt about the effectiveness of this method. At present, the only well-accepted method for

preventing system failures due to residual software errors is through extensive testing. A study for the Netherlands Department of Civil Aviation[43] indicates that since software reliability cannot be accurately determined, reliable software development requires rigorous development procedures and extensive testing. It is generally accepted that a large percentage of the residual errors can be eliminated before testing through strict adherence to a software quality assurance program during software development. An intensive study performed by Lear Siegler Inc. during the development of the Boeing 737/300 Flight Management Computer System (FMCS) indicated that 40% to 50% of the errors detected during the debugging process could have been detected during functional testing of software modules[44]. The Department of Defense standard DoD-STD-2168, discussed by Cooper[45] and Smith[46], outlines a method for ensuring software quality through an active quality assurance program. A similar methodology is also used in IEEE Standard 983, "Guide for Software Quality Assurance Planning" and in a report by the European Space Research and Technology Centre on software quality[47]. It is evident that the use of Computer Aided Software Engineering (CASE) packages throughout the specification, design, development, testing, and maintenance of software can have a great impact on its reliability and lifetime cost. CASE packages can be used to enforce a quality assurance by providing a manageable program throughout the software life cycle.

In summary, there is ample evidence that distributed processing can produce a system that is lower in cost, has higher reliability, and greater flexibility than a centralized processing system. However, there is little more than empirical data available to the systems engineer for the optimal design of a distributed system, partly because design of an embedded, real-time, distributed system is highly application-specific. While a distributed design can be used to achieve a more cost-effective solution, it is often unclear how to optimize the design solution for the greatest benefit. The design is typically balanced between the system cost and the acceptable level of reliability/flexibility. The only proven method of achieving reliability in a distributed system is through hardware redundancy. Several software methods of improving system reliability have been proposed, but their value is questionable. A current trend is toward the development of highly reliable processors that achieve fault tolerance through hardware methods. These processors would provide very high levels of fault tolerance while placing little or no extra burden on the software designers. Software development continues to be the greatest hurdle for distributed systems design, with exhaustive testing the only accepted method for producing reliable software. Since exhaustive testing of a complex software system often requires an inordinate amount of time, the only feasible approach to reliable software development is through a highly structured development process.

Section II discusses the design of GASS, detailing the functions required of the system, the system's design constraints, and the development of an architecture that will satisfy both the requirements and constraints. Section III provides in-depth details of the final GASS design, including a description of the hardware and software paths traversed by a sensor datum. Section IV summarizes the major points of this report. Section V provides recommendations for areas in GASS that require further investigation and recommends future system enhancements.

II. System Design

A. System Functions

Figure 1 illustrates the basic and essential functions that are required of GASS. The primary function of GASS is to collect, format, and time stamp data from the sensors and to record these data onto magnetic tape. The magnetic tape contains all data measured by the sensors, and these data are loaded later onto a home-base computer for analysis. The secondary function of GASS is to provide centralized control and monitoring of its sensors and ancillary equipment. Including control in the functions of GASS tremendously escalates its complexity. However, over 30 devices will be included in GASS, many of which are extremely complicated to operate. Centralizing control and monitoring of all the devices in the system will reduce the manning and training levels required to operate the system. The tertiary function of GASS will be navigation. Since

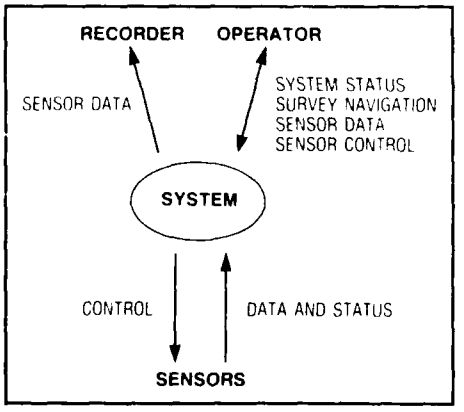

Figure 1. General system functions.

GASS is to be used for surveying, survey tracks must be generated and followed. Furthermore, deviations of the aircraft from the planned track can invalidate the data collected; the system operator must be alerted if the aircraft strays too far from the designated track. GASS navigation functions must include generating and editing survey plans, ascertaining the aircraft's position from its sensors, displaying aircraft position relative to the survey plan to the system operator and the pilot, and keeping a navigation log. The navigation log is a backup, on magnetic media, of the aircraft's track. This information can be used to resume a survey in the event of a system malfunction. A supplementary function of GASS will be data analysis. GASS will have the ability to perform analysis of data previously recorded on magnetic tape, or to analyze data in near real time as it is collected from the sensors.

B. Design Requirements

Typical avionic system requirements involve size, weight, power consumption, and electromagnetic interference considerations. To meet these requirements an avionic system is usually custom-built for a particular application. The original GASS, built by the Applied Physics Laboratory of Johns Hopkins University, consisted of many custom-made devices and interfaces. This design methodology tends to make a system very difficult to maintain and modernize. The need for the current redesign of the system is a direct consequence of the previous system design. The components in the original system can no longer be maintained due to lack of technical support, and the system was not designed to allow for modernization. The primary design constraint put forth by NAVOCEANO for the new GASS is that the system be constructed with a highly modular architecture, using off-the-shelf components if possible. NAVOCEANO desired a system that would be easily reconfigured for different survey missions, but most of all, they desired a system that would allow easy integration of state-of-the-art sensors as they become available.

The mission of GASS will often take it to remote areas of the world, distant from technical support activities. As a consequence, NAVOCEANO desired that the system be designed for maximum reliability. However, since a failure in GASS cannot result in personnel injury or equipment damage, it was not deemed necessary to include extensive fault tolerance in the system. System fault tolerance requires equipment redundancy, which significantly affects the cost and weight of the system. The design sought, then, should improve system reliability through fault detection/isolation and through system availability. The fault detection and isolation capabilities of GASS are discussed in depth by Bourgeois[48]. With advanced fault detection and isolation features, a system will quickly alert the operator of any irregularities. High availability implies that while system operation may be interrupted by failures, the system may be easily and quickly placed back into operation. A high degree of system availability can be achieved through an architecture that allows easy reconfiguration of the system into a degraded operating mode. Thus, reliable system design for GASS will entail a system that will rapidly detect and announce failures, and allow for rapid reconfiguration to a degraded operating mode so that the survey may be continued with minimum impact.

C. System Architecture

1. Number of Processors

One of the first items that must be established about the design of a system of this type is the number of processing elements that will be required. As shown in Figure 2, two extremes are possible: a single processor interfaced to all of the equipment in the system, or a design where each piece of equipment has a processor and all processors are connected to a network. If only the data acquisition tasks are considered, then a single microprocessor should be able to handle the computational load. With approximately 20 sensors and related instruments in GASS, and a maximum sampling rate per sensor of 16 Hz, there are approximately 320 data acquisition tasks must be performed each second. If a 16.7-MHz microprocessor (Motorola MC68020) were to be used for this application, then nearly 50,000 processor cycles would be available for each acquisition task. A crude estimate is five cycles on the average per instruction for the MC68020, so nearly 10,000 instructions per acquisition task are allowed. If only the acquisition tasks were required, then a single MC68020

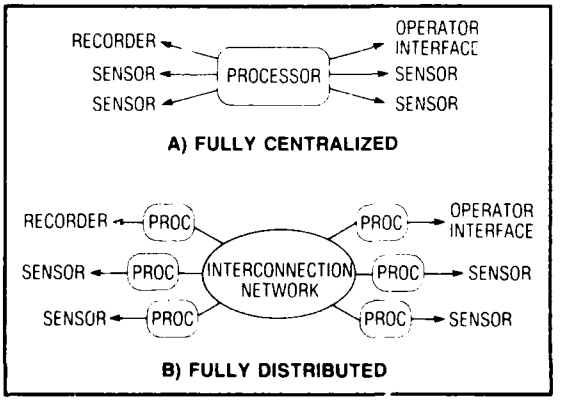

Figure 2. Architectural extremes.

processor would probably suffice. However, in addition to the basic acquisition tasks are the tasks for the tape recorder, navigation processing, operator interfaces, and other system control functions. The addition of these tasks would make it difficult to operate the system with a single processor. Also, the use of a single processor creates a monumental single point failure vulnerability in the system.

The other extreme would be to use a processor for each component in the system, with each processor interfaced to a central bus system. The low cost of microprocessors would make this approach feasible and, if a proper design is used, then multiple microprocessors can reduce the effect on the system of a single failed processor. However, the constraint on GASS for the use of off-the-shelf components restricts this approach. A processor per instrument is most attractive if a custom "black box" (typically a single circuit card) can be built that will interface the component directly to the central data bus. Use of off-the-shelf components, however, would require a separate microprocessor card, an interface card, and a card cage for each device. The result is that more circuit cards would be required, and the overall system power requirement would be greater. As a consequence, the decision was made to use "several" microprocessors, somewhere between the two extremes. The exact number of processors needed depends mainly upon how and where they are to be used in the system, as discussed in the next section.

2. Centralized vs. Distributed Hardware

With the choice to use multiple processors made, a decision must be made about how these processors will be arranged in the system. Two general methods are available: centralized or distributed. A centralized system offers the advantage that a single card cage can be used. This system allows for higher communication rates between processors on a backplane bus system, the elimination of the hardware and software required to network distributed processors, and a reduction in the number of power supplies required for the system. The disadvantages of a centralized system include single point failure vulnerability and a more severe wiring problem. With a centralized system the interface cables must be run from each sensor to the computer system. This configuration results in much more weight and bulk over that which would be required for a network-connected, distributed system. This problem has already caused many high-performance fighter aircraft manufacturing firms to seriously consider using distributed systems. The single point failure disadvantage is, without question, the single greatest problem with centralized systems. Not only can the entire system be halted by a failure of the central computer's power supply, but failures in remote parts of the system can disrupt operations. An extreme example of this problem has been displayed by the Main Engine Control System for the FFG7 Class (Fast Frigate with guided missiles) of naval ships, designed by the General Electric Ground Systems Division. The system is used to control the operation of the two main propulsion gas turbine engines and their ancillary equipment. A centralized system is used, with all sensor and actuator signals routed to the central computer. Component failures remote to the computer can cause unwanted voltages to appear on the component's return signal lead, which is common to the entire system. At best, the result would be an inoperative system. Unfortunately the result would typically be erratic and undesirable system operation.

Distributed systems can nearly eliminate the problem of single point failures. In a distributed system the tasks performed by the system are distributed among several processors, called "nodes," interconnected by a network. Network interfaces are available that allow a system to completely disregard faulty nodes. If the system is designed to reduce inter-node dependencies, then the failure of a single processor can have little or no effect on the rest of the system. A distributed system reduces the wiring bulk by placing the processing elements nearer to the remote system components, and the processing elements can be connected with either a single twisted pair of wires, a coax cable, or a fiber-optic cable. This wiring configuration also reduces a system's vulnerability to electromagnetic interference.

A distributed system increases availability in that a single faulty node will not affect the rest of the system. Furthermore, if the system is so designed, then the tasks of a faulty node can be redistributed to the functional nodes. Disadvantages of a distributed system, in light of the requirement for off-the-shelf components, are that a separate card cage, each with its own power supply and network interface card, must be used for each remote processing node. The decision was made to use a distributed design for GASS, because of its reliability aspects. To reduce the cost incurred by the support equipment required at each node, only eight nodes will be used. Six of these nodes will interface to sensors, the seventh node will be used for the operator and tape recorder interfaces, and the eighth node will handle the data analysis functions. The six sensor nodes will be designed to provide the computational power needed by the current sensor complement, and to allow addition of more sensors. With six sensor nodes, there will be an approximate loading of only three devices per node.

This number leaves ample processing power for the task of bus communications, some minor data processing, and the addition of more sensors.

3. Local Area Network

Since the system will be constructed using a distributed architecture, a communications media must be chosen to interconnect the processors. The IEEE-488 standard, or general-purpose interface bus (GPIB), would seem the natural choice for this application. However, the GPIB poses severe limitations that make the use of a network-type communications system much more attractive. The GPIB has a limitation of 15 devices per bus, and the maximum length of cables used for a bus must not exceed 20 m[49]. Also, the GPIB cable is much heavier and bulkier than a single twisted pair or a coax cable. The GPIB cannot be easily upgraded to a faster communication media; a coax network could easily be upgraded to a faster fiber-optic network. Because of the GPIB limitations, the decision was made to use a network based on a serial data bus to interconnect the processor nodes.

Three fundamental local area network (LAN) configurations are practical for use in an embedded real-time avionics system: the ring, the star, and the bus (Fig. 3). In the ring bus, which is commonly unidirectional, messages are circulated around the ring until they reach the target node(s). This protocol greatly simplifies message routing; and, since more than one message can be in transit, very high data rates can be achieved[50]. The ring bus is well adapted for systems consisting of highly autonomous nodes. Since all the sensor nodes in GASS must communicate with the single node that has the tape recorder and the operator interface, the nodes of GASS cannot be considered autonomous. Therefore, the ring bus is not an efficient choice for this application. The star bus, with the operator/recorder node of GASS placed at the center, would seem the most natural choice for this system. The advantage of a star configuration is that it can achieve the highest data rates of all the possible architectures. The disadvantage of the star configuration is that the center node would require an interface card for each remote node. The bus configuration offers a system with less hardware but slower data rates, since several nodes must share the same communication media. The bus is the typical choice for avionics systems because of the savings in hardware[13]. A single bus architecture was selected as the economical choice for the GASS network, since each node will have the ability to buffer collected sensor data between bus transmission periods.

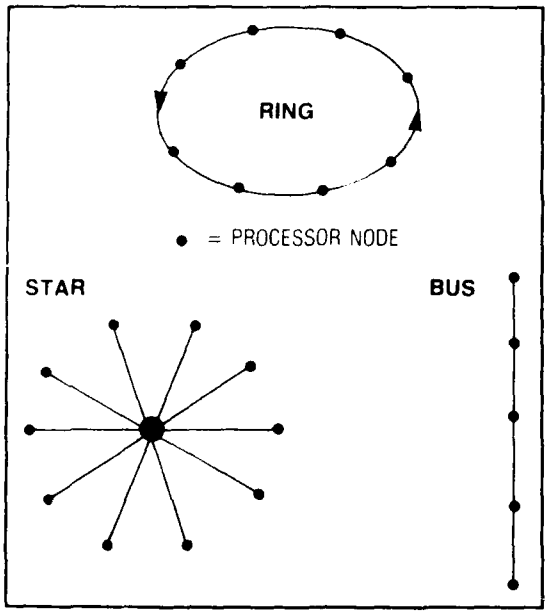

Figure 3. Standard network topologies.

When a network is used for communications, a Media Access Protocol (MAP) must be established. There are two fundamental MAP's, contention and noncontention[50]. With a contention MAP, also known as random-access MAP, all nodes have equal access rights to the network. The most commonly known network of this type is Xerox's Ethernet. A node gains access to the network by listening; if the network is silent, then it will proceed to transmit. Since there are propagation delays across the wiring in a network, two nodes may begin transmissions simultaneously, resulting in a "collision." Collision is the greatest disadvantage of a contention MAP, and its occurrence makes this protocol unsuitable for real-time applications. Current research, however, indicates that the problems of collision in random-access networks may soon be remedied[51, 52].

With a noncontention MAP, some method is used to ensure that only one node is transmitting on the network at any given time. The two common methods for implementing a noncontention MAP are bus controller and token passing. With a bus controller MAP, a single node is given the responsibility to control all network communications. Other nodes are instructed to transmit data packets by the controller node. The greatest advantage of this scheme is its inherent simplicity of implementation. The disadvantages are that it does not make maximum use of the bandwidth of the communication media, and it presents a single point failure vulnerability. The time-slot implementation is the typically used controller scheme, where each remote node is allotted a period of time for transmission by the

controller. Inefficiency exists with this implementation since a node may be granted its time-slot even if it has no data to transmit. The token-passing MAP has received much attention in recent years[53-60], and it provides for more efficient use of the network medium. With a token passing MAP a token is circulated from node to node throughout the network. The node that possesses the token is allowed to transmit on the bus and, when it no longer needs network access, it is responsible for passing the token to the next node. The noncontention controller MAP was chosen for GASS because of its simplicity of implementation and because the system architecture is such that all sensor nodes will communicate with the operator/recorder node but not with each other. The problem of single point failure vulnerability presented by this MAP can only be resolved by using an additional node with a tape recorder as an online backup for the controller node. The cost in terms of expense, weight, and size make this solution unacceptable for GASS. As discussed in the next section, the analysis node will be used as an off-line backup for the controller node.

4. Physical and Functional Partitioning of Software

The hardware design concept consists of eight processing nodes interconnected by a bus network that uses a controller MAP. Six of the nodes will be used for sensors. The seventh node will be used for the operator interface and the magnetic tape recorder. The eighth node will contain the data analysis functions. Since all sensor nodes need to communicate with the operator/recorder node, but not with each other, the network controller function will be located in the operator/recorder node. All that remains to be defined for the system's architecture is the placement of the various software tasks required to operate the system.

The software tasks required for this system include bus interface tasks at each node, operator interface tasks (keyboard, displays, etc.), and a tape recorder task at the operator/recorder node, sensor control and acquisition tasks, navigation tasks, and data analysis tasks. To maximize real-time performance, the software tasks should be physically and functionally partitioned. In other words, the software should be designed to be modular in form. To meet this goal all sensor tasks are completely isolated to the sensor nodes. They will be designed such that all sensors appear to be identical from the perspective of the operator/recorder node and from the bus interface task in each node. More specifically, the individual sensor task in a sensor node will be the only software that is aware of the actual details of operation of its particular sensor. To the rest of the system, the sensor will have only the basic functions of sampling or not sampling, sampling rate, and initialization. By placing all of the "intelligence" to operate a sensor only at the sensor's node, network communications are decreased and the speed of the sensor task is increased, since it is not dependent upon the network. Furthermore, data from all sensors will be packaged identically, eliminating special handling requirements insofar as the bus interface and tape recorder tasks are concerned. This software approach also offers the advantage that all bus interface tasks will be identical. This method will reduce the cost of software development and, since the same module will be used by several developers in different nodes, it provides more thorough debugging of the software module. Another advantage is that the highly modular sensor tasks may be easily relocated to different nodes within the system.

To reduce the possibility of a bottleneck in the network, the decision was made to use two microprocessors in the operator/recorder node. One processor would be used strictly for the bus interface and the recorder interface to minimize the time required to pass data from the network to the tape recorder. The real-time processor would handle all communications on the network, and would pass the required control and data information between the network and the operator interface. The second processor would be used for the operator interface, which is a less time-critical function. This processor is the logical location for the navigation task, which is predominantly an off-line task, requiring only occasional position data from the sensors. Since data analysis tasks are typically computationally intensive, a separate processor, isolated from the real-time elements of the system, should be used. The need to isolate potentially slow software tasks from the rest of the system was the primary reason for using a separate node for the data analysis tasks. Since data analysis will be performed on a separate node, this node must also include the hardware and the software tasks to receive real-time data from the network or to read data tapes recorded by the operator/recorder node. The data analysis node will thus require much of the same hardware and software as the operator/recorder node, so these two nodes were made using identical hardware. Data analysis is a nonvital task in GASS and, in the event of a failure of the operator/recorder node, this design scheme would allow the data analysis node to be used for control of the system.

The resulting architecture, illustrated in Figure 4, has the software tasks physically and functionally partitioned. The highest speed tasks, those of sensor control and data acquisition, are located only in the six sensor nodes. The next highest speed requirement is that of moving the data from the sensors to the tape recorder.

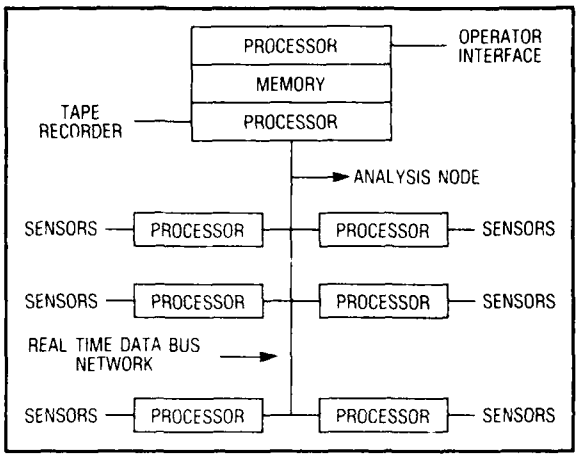

Figure 4. GASS architecture.

This requirement is handled by bus interface tasks in each sensor node, and the bus and tape recorder tasks are handled on the real-time processor in the operator/recorder node. The slowest tasks, operator interface and navigation, are handled in the second operator/recorder node processor, which is physically separated from the real-time processor by a global memory area. All communications between the two processors in this node will be handled through the global memory, reducing the load on the real-time processor. The data analysis node will function independently from the rest of the system, and will "listen" to the system's network to receive real-time data from the sensors.

D. System Building Blocks

1. Microprocessor/Backplane

The choice for the backplane bus system was largely driven by the requirement to use off-the-shelf components, and the desire to use a 32-bit data and address bus. The oldest 32-bit microprocessor standard bus is the Motorola VME bus system introduced in 1981 (also known as IEEE Standard P1014)[61]. The VME bus had the greatest market support at the time of the GASS design and was chosen as the microprocessor backplane bus system. As a consequence, despite the diverse array of sensors used, only 3 of the 17 interface cards in the system had to be custom-made. Since the VME backplane was selected, a natural choice for a microprocessor was the Motorola MC68020. The MC68020 is a state-of-the-art microprocessor that supports multitasking, and the 68000 family of processors have been used in aerospace applications by Plessey Avionics[62] and Draper Laboratory[17, 24, 63]. It was the second most preferred microprocessor (after the MIL-STD-750A) of Ada development tool manufacturers[64].

2. Local Area Network Medium

The LAN medium selected by NAVOCEANO was the Military Standard 1553B serial bus, the data bus of choice for military avionic systems (and, thus, a large industrial backing). It supports 1-Mbps operation, transformer isolation of nodes, and offers the feature of dual-redundant communication channels. A possible alternate bus would be the Avionics Standard Communication Bus (ASCB)[65]. ASCB was developed by Sperry Corporation and is supported by the General Aviation Manufacturers Association (GAMA). It is also a dual-redundant bus system for avionics applications, and supports 0.67 Mbps operation at a reduced cost. The chief advantages of the 1553B bus over the ASCB is that 1553B has broader industrial support, and fiber-optic upgrades for 1553B systems are already available[66].

3. Operating Systems

Unix (Motorola V.3) was chosen as the operating system for the operator interface processor. Unix is a well-established, multitasking operating system with which the system's designers were well acquainted. A full-capability operating system is needed for the operator interface processor, since it is responsible for standard microcomputer tasks, including hard/floppy disk drive control, display control, and printer control, as well as its GASS specific tasks. For the real-time processor and the processors in the sensor nodes, it was desired to have a streamlined operating system that was smaller and faster. For this application the pSOS operating system was selected. Manufactured by Software Components Group, pSOS is a real-time multitasking operating system that is tailored for the Motorola 68000 family of microprocessors.

4. Programming Language

Although some assembly-language programming was required for this type of system, the design goal was to use a high-level programming language whenever possible. High-level languages permit easier system development and provide much easier system modification and modernization. The language chosen for this system was Kernighan and Ritchie's "C." C is an excellent language for developing a system that involves a significant amount of input/output processing and special-purpose equipment interfacing. A possible contender for the GASS programming language choice was Ada, the Department of Defense's (DoD) standard programming language for real-time embedded systems. Even though Ada has been used by many DoD contractors and is selected as the programming language for the Space Station,

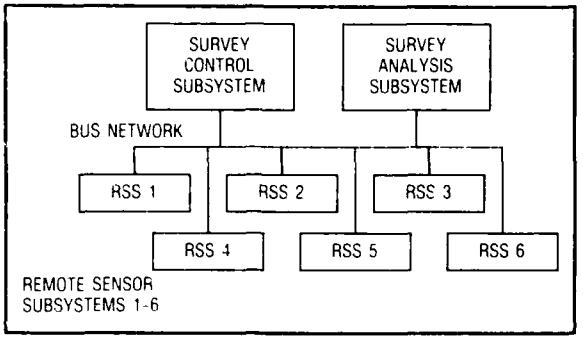

Figure 5. GASS subsystems.

Columbus[67], Ada has only recently been considered a mature programming language[62, 64, 68, 69]. Reported problems with Ada include difficulty in obtaining compilers for microprocessors and differences in interpreting the language between validated compilers; also the code produced is too large and slow for embedded systems. Ada was considered to be inappropriate for use in GASS due to the lack of familiarity by the design team and to the difficulties reported by many large government contractors.

III. System Details

A. Hardware

GASS is comprised of three major subsystems as illustrated in Figure 5. These subsystems are the Survey Control Subsystem (SCS); the Survey Analysis Subsystem (SAS); and the Remote Sensor Subsystem (RSS), which is comprised of six remote sensor systems (RSS1 through RSS6).

The SCS is the central control point of GASS. This subsystem collects data from the RSS, stores the collected data on magnetic tape, and displays the necessary survey status information for the GASS operator and the Project MAGNET aircraft pilot. The SCS is designed to collect and store all system data at rates of 1, 2, 4, 8, or 16 times a second.

The SAS provides the capability to perform in-flight analysis on the data being collected and to display the results graphically. Each of the six remote sensor systems are comprised of a remote sensor controller (RSC), sensors, and sensor supporting instruments. The RSCs in the remote sensor systems are identical, with the exception of the special interface cards required for the various sensors. The primary communications bus for GASS is a dual-redundant MIL-STD-1553B serial data bus that connects to the SCS, the SAS, and each RSC. Other data buses are used within the SCS, the SAS, and each remote sensor system as required by their associated equipment.

1. SCS/SAS and the 1553 Data Bus

The SCS and SAS subsystems are illustrated in Figure 6. The SCS consists of the survey control computer, a 9-track tape recorder, an operator console, a graphics display, the project display, the navigation display, the pilot display, and a printer. The survey

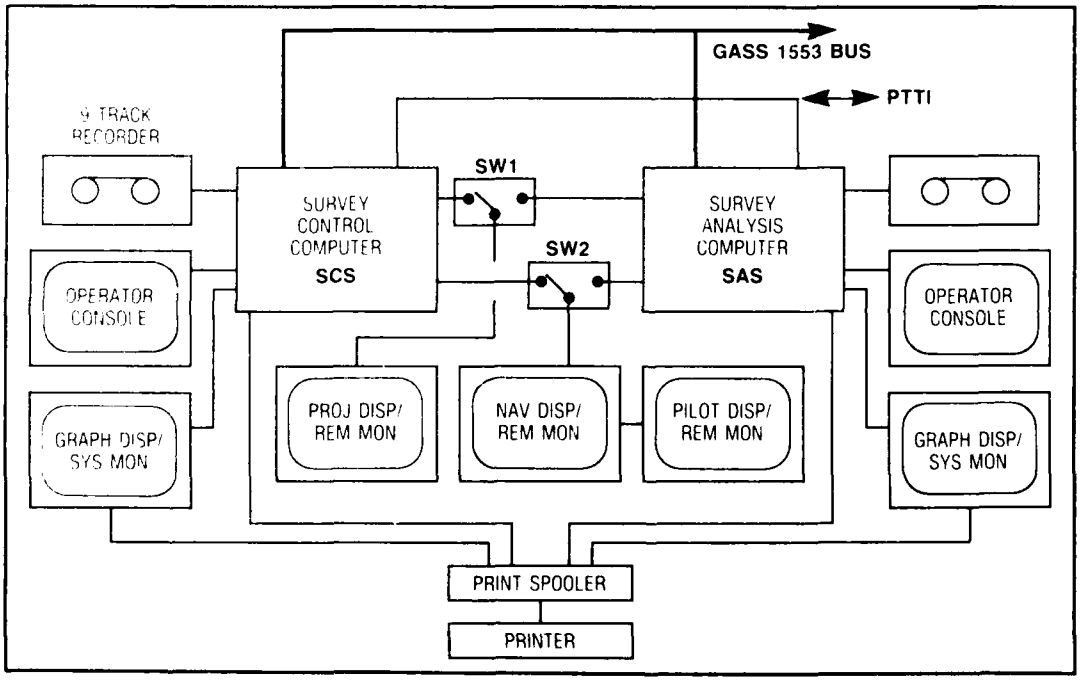

Figure 6. Survey control and analysis subsystems.

control computer is a Plessey CS-10 computer system, with a Plessey 68-22M CPU card and a Plessey 68-25M CPU card. The 68-22M CPU card is used for the operator interface and system control functions. The 68-25M CPU card provides an interface between the RSS, the tape drive, and the operator. The 68-25M card interface to the operator is via a global memory card that is accessible by the 68-22M CPU card. Both CPU cards are based on a Motorola 68020 microprocessor with a Motorola 68881 co-processor. The operator, graphics, project, and navigation displays are Tektronics SF4208 graphics terminals. The pilot display is an RGB (red-green-blue) display mounted in the cockpit of the aircraft. This display provides the pilot with the survey route and other related information. The primary function of the SCS is to collect data from the RSS over the 1553 data bus and to store this data on tape. The SCS also provides survey navigation information, GASS equipment status, backup system timing, and operator system control functions.

The SAS consists of the survey analysis computer, an operator console, a graphics display, and a 9-track tape recorder. The primary purpose of the SAS is to provide the operator access to the GASS survey data base so that in-flight data analysis may be performed. As shown in Figure 6, switches SW1 and SW2 allow either the SCS or the SAS to operate the project, navigation, and pilot displays. The SAS hardware is identical to the SCS, enabling the SAS to operate GASS in the event of a catastrophic failure of the SCS.

The MIL-STD-1553B data bus is a serial data bus with a 1-MHz maximum bit rate. The dual-redundant configuration of this data bus is used in GASS. This configuration provides a dual path for all communications between the SCS/SAS and the RSCs. In the event that one of the redundant buses fails, GASS will automatically switch to the secondary bus without loss of data. The 1553B data bus also provides the feature that a failure of an individual processor connected to the bus will not disable the bus.

2. System Timing

An accurate time reference for GASS is absolutely crucial to enable post-processing of the geophysical data collected. Accurate time references are contained in all processing units: the SCS, the SAS, and each RSC. Since each of these eight units contains its own timing device, some method of assuring synchronization between all devices is required. The Precise Time and Time Interval (PTTI) interface provides time synchronization between all processing systems in GASS. PTTI provides time synchronization once every minute to the SCS, the SAS, and each RSC. This is accomplished through three lines: the binary coded decimal (BCD) line, the pulse per minute (PPM) line, and the timing fault line. The BCD line provides the current time stamp encoded in binary coded decimal. The PPM line provides the timing mark for the time issued by the BCD line. The timing fault line indicates the source of the PTTI signals. During normal operations the PTTI signals are derived from the global positioning system (GPS) through the PTTI interface custom-built by NORDA. In the event that the GPS receiver fails to produce the PTTI signals, the timing fault line is asserted by the SCS, and the SCS takes control of the time synchronization function. In this mode the SCS sends the time stamp to the SAS and each RSC over the 1553 serial data bus, and then issues the timing mark over the PPM line.

3. Remote Sensor Subsystem

The RSS consists of all six RSCs and their associated sensors. The RSCs are custom-built VME chassis manufactured by NORDA. They provide the power supply and framework to mount the processor and interface cards required to operate an individual RSS. Each RSC contains a Plessey 68-25M CPU card, an SCI Corporation 1553B serial interface card, and any interface cards required by the sensors of a particular RSS. Sensor interface cards include custom parallel, custom serial, 1553B serial, ARINC-419, synchro, GPIB, and RS422 interfaces. The following paragraphs describe the components of each RSS.

RSS1 (Fig. 7) contains RSC1, a Rosemount 1501AT precision barometric altimeter, a Rockwell GPS-3A receiver, and the NORDA PTTI interface. The altimeter provides altitude in feet to the RSC through a parallel interface. The Rosemount altimeter is the most accurate barometric source for altitude on the aircraft. The GPS receiver provides latitude, longitude, altitude,

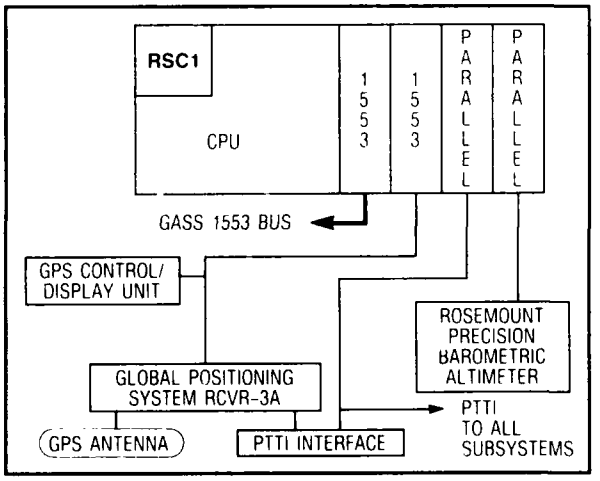

Figure 7. RSS1 interface diagram.

and heading to the RSC through an independent 1553 serial bus. The GPS receiver also provides the signals required by the PTTI interface to provide system synchronization. The GPS receiver provides the most accurate aircraft position information in the system. It calculates aircraft position through communications with four satellites (normal operation) or through communications with three or less satellites and a barometric altitude input. Barometric altitude is supplied by the SCS over the 1553 data bus to RSC1.

RSS2 (Fig. 8) contains RSC2, an AAU-21 barometric altimeter, and two Litton-72 inertial navigation systems (INS). The AAU-21 altimeter provides altitude in feet to RSC2 through a parallel interface. Each Litton INS provides heading, pitch, roll, latitude, longitude, north velocity, east velocity, ground speed, and drift angle to RSC2 through an ARINC-419 serial interface and a Synchro interface. These devices are part of the aircraft navigation system and are not controlled by GASS. The barometric altitude data required by the Litton INSs is provided by the aircraft's systems, independent of GASS.

RSS3 (Fig. 9) contains RSC3, a Texas Instruments ASQ-81 scalar magnetometer, two Hewlett-Packard HP3570B frequency counters, a Gould RS3200 strip-chart recorder, and an RMS Instruments automatic aeromagnetic digital compensator. The ASQ-81 magnetometer detects local magnetic field intensity using a sensor head in the tail of the aircraft. It produces two signals: the Larmor frequency and the bandpass output. The Larmor frequency is a frequency between 0.6 and 2.2 MHz that is proportional to magnetic field intensity. The Larmor frequency is measured by the HP5370B frequency counters and input to RSC3 over an HPIB data bus. Two frequency counters were required to provide the 16-Hz maximum sampling rate required of the system. An individual HP5370B frequency counter is capable of a maximum sampling rate of only 10 Hz in this configuration. The bandpass output is a signal that is proportional to the rate of change of the local magnetic field intensity. This signal is input to RSC3 through an analog to digital converter card. The strip-chart recorder provides the GASS operator with a quick-look display of both outputs of the scalar magnetometer. The digital compensator provides magnetically compensated total field and gradient signals to RSC3 through a parallel interface. It derives these signals from the ASQ-81 scalar magnetometer's Larmor frequency signal and the X, Y, and Z axis magnetic intensities from the vector magnetometer in RSS4.

RSS4 (Fig. 10) contains RSC4, a Honeywell H-423 ring laser gyro (RLG), a Honeywell electronically suspended gyro (ESG), a NAROD vector magnetometer, and three HP3457A multimeters. The RLG provides latitude, longitude, pitch, roll, and true heading to RSC4 through an independent 1553B serial

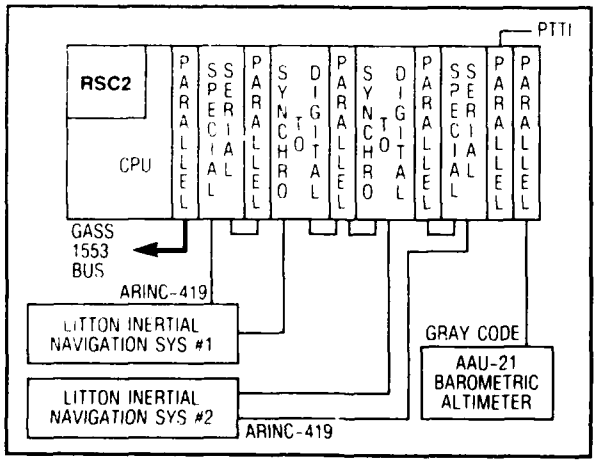

Figure 8. RSS2 interface diagram.

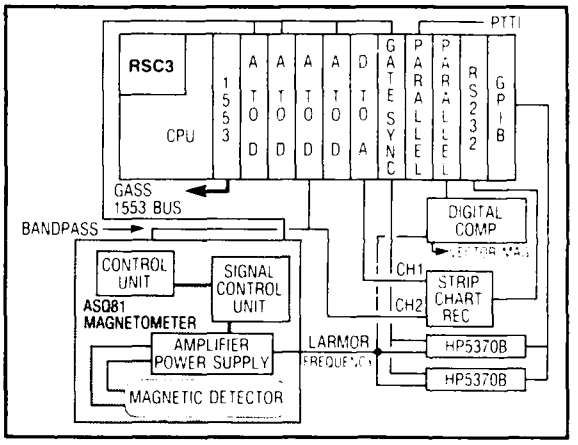

Figure 9. RSS3 interface diagram.

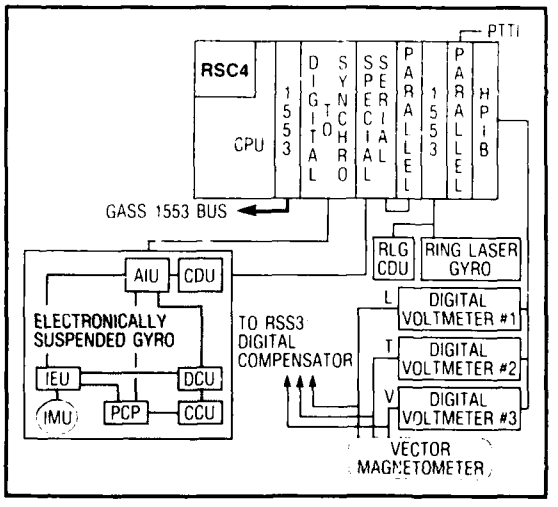

Figure 10. RSS4 interface diagram.

data bus. The ESG provides latitude, longitude, pitch, roll, and true heading to RSC4 through a special serial interface. RSC4 provides both the RLG and the ESG with barometric altitude supplied by the SCS. The vector magnetometer produces three voltages corresponding to the orthogonal magnetic field vector intensities. These voltages are measured with the digital multimeters and sent to RSC4 over an HPIB data bus. These voltages are also supplied to the digital compensator in RSS3.

RSS5 (Fig. 11) contains RSC5, an OPTECH 501-A laser altimeter, and a Honeywell APN-222 radar altimeter. The laser altimeter provides altitude in meters to RSC5 through a parallel interface. The radar altimeter provides altitude in feet to RSC5, also through a parallel interface. The laser altimeter is the most accurate source of altitude in GASS for low altitudes, and the radar altimeter is the most accurate for high altitudes.

RSS6 (Fig. 12) contains RSC6, the Hydrographic Airborne Laser System (HALS), and the Airborne Multispectral Pushbroom System (AMPS). HALS is under development by NORDA and the Naval Air Development Center (NADC) and is a laser-based system for determining shallow-water depth[70]. AMPS is under development by Lockheed through NASA and is a system that determines shallow-water depths using multispectral images of the ocean surface. These systems will collect and store their data independently from GASS, and they will have their own operator on the aircraft. GASS provides HALS and AMPS with a time stamp through a parallel interface, and with latitude, longitude, height, true heading, pitch, and roll through a serial interface.

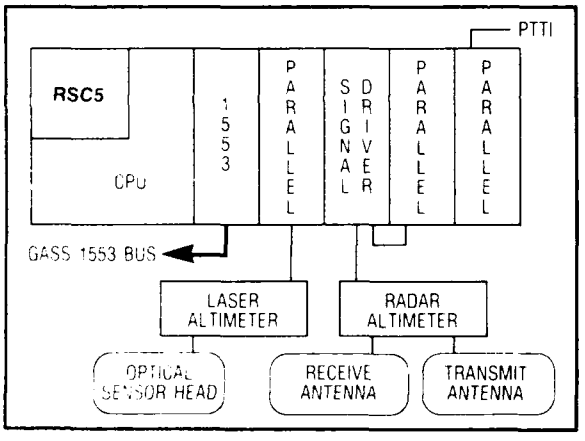

Figure 11. RSS5 interface diagram.

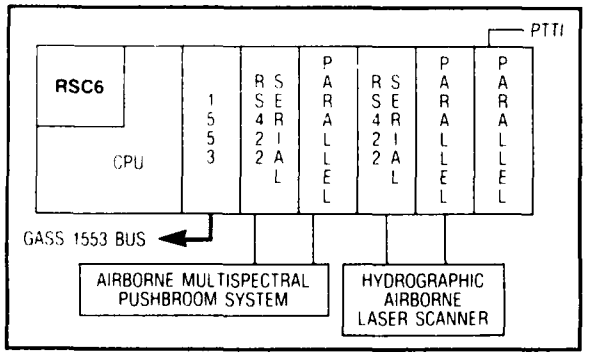

Figure 12. RSS6 interface diagram.

B. GASS Software

Figure 13 illustrates the GASS software distribution. GASS software is functionally and physically distributed among the processors in the SCS, the SAS, and the RSCs. All real-time or near-real-time functions are handled by the 68-25M CPU cards in the SCS, the SAS, and the RSCs under a Software Components Group Inc. operating system called pSOS. All nonreal-time functions are handled by the 68-22M CPU cards in the SCS and the SAS under a Motorola Unix V.3 operating system. The software is functionally distributed into the following areas: Survey Control System, SCS Real Time Front End (RTFE), Survey Analysis System, SAS RTFE, and the RSCs.

The SCS software is responsible for the operator interface, system control, navigation, displays, and printer functions. The SCS RTFE software is responsible for the 1553 communication, data handling, message processing, tape drive control, and timekeeping functions. The SAS software is similar to that of the SCS, but it does not incorporate the system control and navigation functions. The SAS software includes graphics and data base functions not in the SCS software. The SAS RTFE software is almost

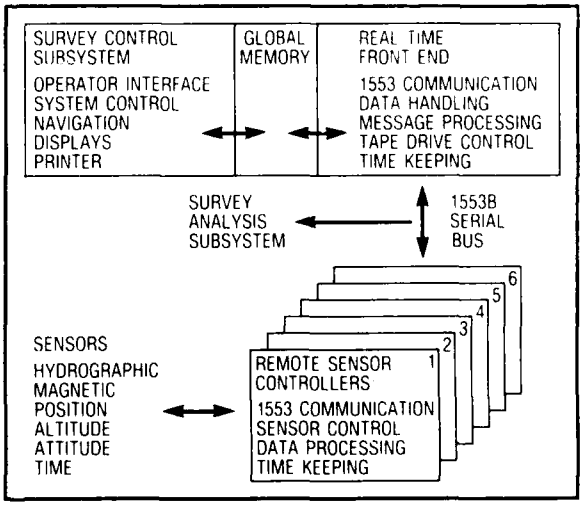

Figure 13. GASS software distribution.

identical to the SCS software. The RSC software is responsible for time keeping and 1553B data bus communications for all RSCs. It also performs the functions of sensor control and sensor data pre-processing for the sensors in a particular RSS.

The software in the six RSCs is physically separated from the RTFEs in the SCS and SAS by the 1553B serial bus. The software in the SCS and SAS is physically separated from the respective RTFEs by the SCS and the SAS Global Memory Cards, with the exception of a "mailbox" interrupt line. The SCS and SAS software is stored on hard disk and is loaded into RAM for operation. Both of the Plessey CS-10 computer systems contain the software for the SCS and the SAS, and each may be booted to perform either function. The software for the SCS RTFE, the SAS RTFE, and the RSCs is contained in ROM in the respective 68-25M CPU cards, and is loaded into RAM for operation. The ROM used for the SCS and SAS RTFE contains the software for both, allowing either CS-10 computer system to operate as either the SCS or the SAS. The ROMs for the RSCs contain the software for all six RSCs. A jumper on a parallel port of the RSC is used to inform the RSC boot software as to which RSC it is in upon power-up. The boot software will load the software for that RSC into RAM.

1. Survey Control Subsystem Software

The SCS software (Fig. 14) consists of the navigation history, navigation control, display control, data print, mailbox control, and man-machine interface tasks. The navigation history task keeps a record of the survey's navigation information on the computer's floppy drive. This information is used to quickly resume a survey should an SCS failure occur. The navigation control task is responsible for computing the deviation from the actual aircraft's track to the desired survey track. The resulting information is displayed on the navigation and pilot displays. The display control task is responsible for driving the non-interactive navigation, project, and graphics displays (the pilot display is a slave under the control of the navigation display). The data for these displays are obtained from the SCS global memory. The data print task creates printouts of data and system parameters, as directed by the operator. The SCS mailbox control task provides for the interface between the SCS software and the SCS RTFE software. The man-machine interface task controls all interactions with the operator via the operator console. Its functions include on-line help, configuration and control of systems sensors, generation and editing of survey tracks, display configuration, special events log, and print control. With the exception of the mailbox interrupt line, all communications between the SCS software and the SCS RTFE software are via the SCS global memory.

2. SCS Real-Time Front-End Software

The SCS RTFE, software (Fig. 15) consists of the 1553 bus manager, message processor, tape manager, and time tasks. The 1553 bus manager task in the SCS RTFE controls all communications on the 1553B dual-redundant serial data bus. It operates the SCS's SCI 1553B interface card in the bus controller mode, polling each RSC for status and data. The message processor

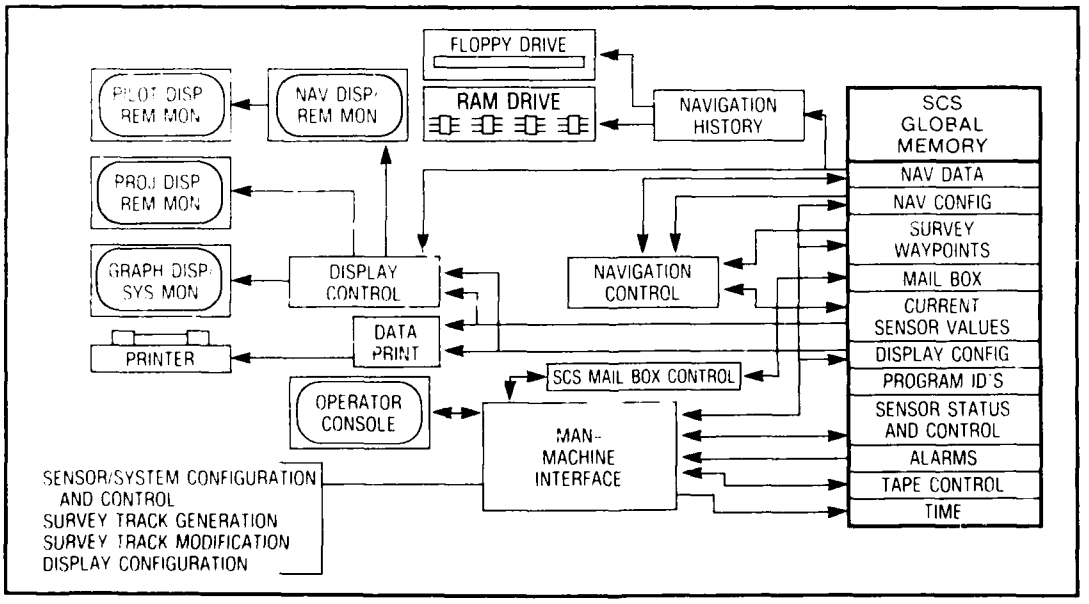

Figure 14. Survey control subsystem software (UNIX operating system).

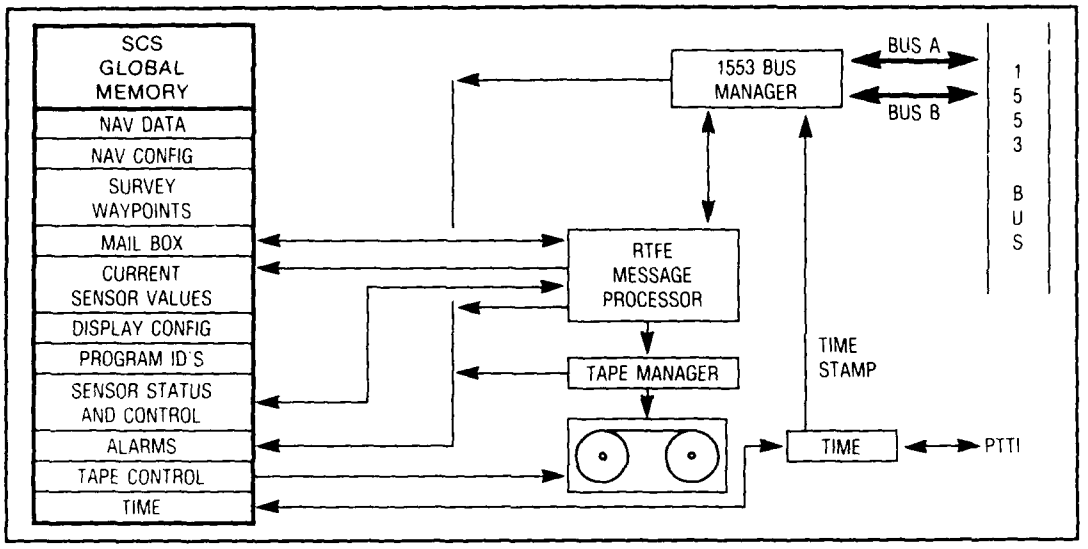

Figure 15. Real time front end software (pSOS operating system).

task handles all communications to and from the 1553 bus manager task. It handles the formatting and routing of all messages and data. It sends sensor control commands to the bus manager, receiving sensor status messages and sensor data from the bus manager, and for routing sensor data to the tape manager task and the SCS global memory. The tape manager task formats all data received from the message processor and for operation of the magnetic tape drive. The time task in the SCS updates the RTFE's processor clock using the PTTI signal. This task also provides the current system time to the operator via the SCS global memory. The SCS RTFE time task serves as the backup for the PTTI signal in GASS. If this task recognizes a loss of the PTTI signal, then it will take over the system timing function by asserting the PTTI fault line, generating timing sync pulses over the PTTI interface and sending the current time stamp to the bus manager for transmission to all RSCs.

3. Remote Sensor Controller Software

The remote sensor controller software (Fig. 16) consists of the 1553 bus manager, message processor, time manager, and sensor control tasks. The bus manager task in the RSCs buffers and transmits the data and messages received from the message processor. It also receives sensor control information from the 1553 bus and passes this to the message processor. The RSC bus manager task operates the SCI 1553B interface card in the remote terminal mode. The message processor task processes all messages between the sensor control tasks, the bus manager task, and the time manager task. The time manager task updates the RSC processor's clock using the PTTI signals. If the PTTI fails, then the

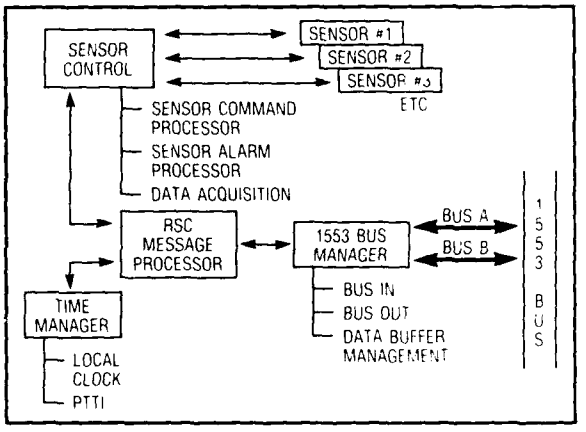

Figure 16. Remote sensor controller software (pSOS operating system).

time manager task will issue an alarm to the message processor for transmission to the SCS. The RSC software contains a separate sensor control task for every sensor or instrument in the particular RSC. Each sensor control task is responsible for sensor operation and control (via commands from the SCS), sensor alarm processing, data acquisition, and data checking/formatting.

4. Survey Analysis System and SAS RTFE Software

The SAS RTFE software (Fig. 17) is similar to the SCS RTFE software, except that the tape manager task is included in the message processor task, and the bus manager operates the SCI 1553B interface card in the bus analyzer mode. The bus analyzer mode allows only the SAS to monitor the 1553B data bus. A separate tape

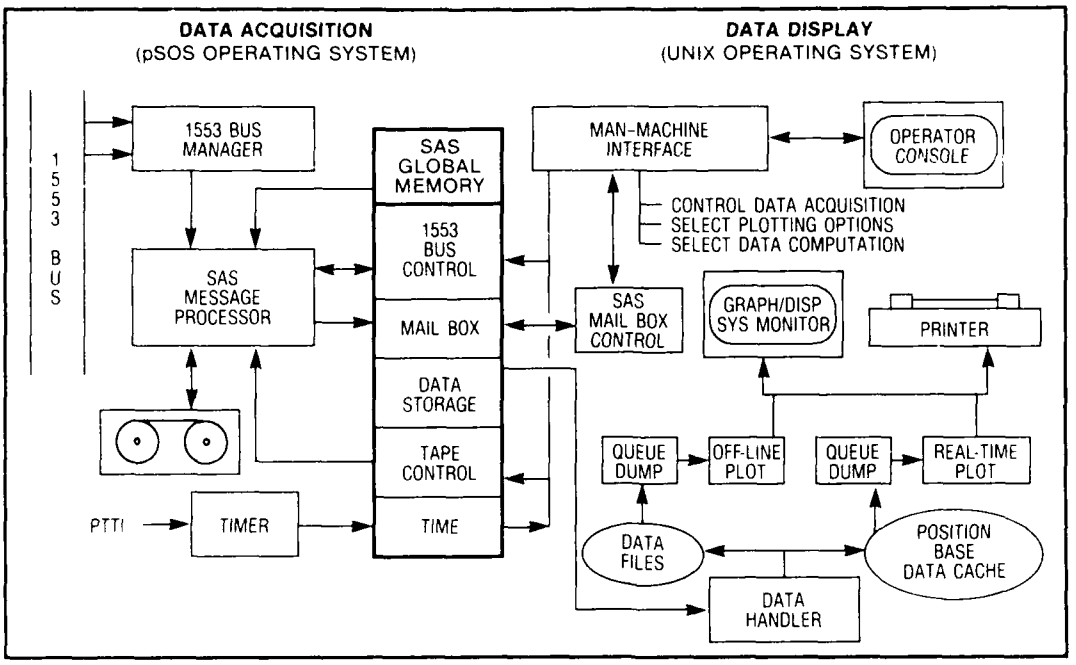

Figure 17. Survey analysis subsystem software.

manager task is not needed, since the SAS will only retrieve data from the tape recorder and will not send data to the recorder. The SAS software consists of the man-machine interface, SAS mailbox control, data handler, que dump, and plot tasks. As with the SCS, the man-machine interface provides all interface between the operator and the SAS. This task allows the operator to control data acquisition from the 1553B bus and tape drive, to select plotting options, and to select data computations. The SAS mailbox control task provides the interface between the SAS software and the SAS RTFE software. The plot tasks can generate real-time plots from the data base or from off-line plots from data stored in files. The plots created can be displayed on the graphics display or on the printer. The data handler task routes data from the SAS RTFE to either the data base or to disk files. The que dump task routes the operator-specified data from either the disk files or the data base to the plot tasks.

C. GASS Data Flow

Figure 18 illustrates the typical flow path of data through GASS. Data originate at an individual sensor and are sent to an RSC via the sensor's interface card. The sensor interface card converts the data from the sensor into a format that can be used by the RSC's 68-25M CPU card. The sensor manager task receives the data from the sensor interface card and sends it to the message processor task. The message processor task properly formats the data and sends it to the bus manager task. The bus manager collects the data from all sensors into a buffer. When the RSC is polled by the SCS, the bus manager sends all of the data currently in its buffer to the 1553B interface card for transmission on the 1553B data bus. The SCS RTFE bus manager task receives the data transmitted on the bus by the RSC from the SCS's 1553B interface card. The bus manager task passes the data to the SCS RTFE message processor task. The message processor routes all the sensor data to the tape manager task and to the sensor data area in the SCS global memory card. The tape manager task formats the data and sends them to the Pertec interface card, which in turn sends the data to the tape drive for recording. The display control task, located in the SCS CPU card, retrieves the data placed in the SCS global memory card by the SCS RTFE CPU card, formats it, and sends it to the displays via a serial interface.

IV. Summary

This study evaluates and applies the state of the art of distributed real-time system design. The knowledge base for distributed design is largely empirical, and few metrics are available for evaluating the economics of a particular design. The significant gains of a distributed design over a centralized design are an increase in reliability, flexibility, and expandability. Software development is the most difficult and the most expensive part of distributed systems development. While distributed systems can realize a savings in hardware, the cost of the software could easily offset this savings if the system design is too complex. It is important to

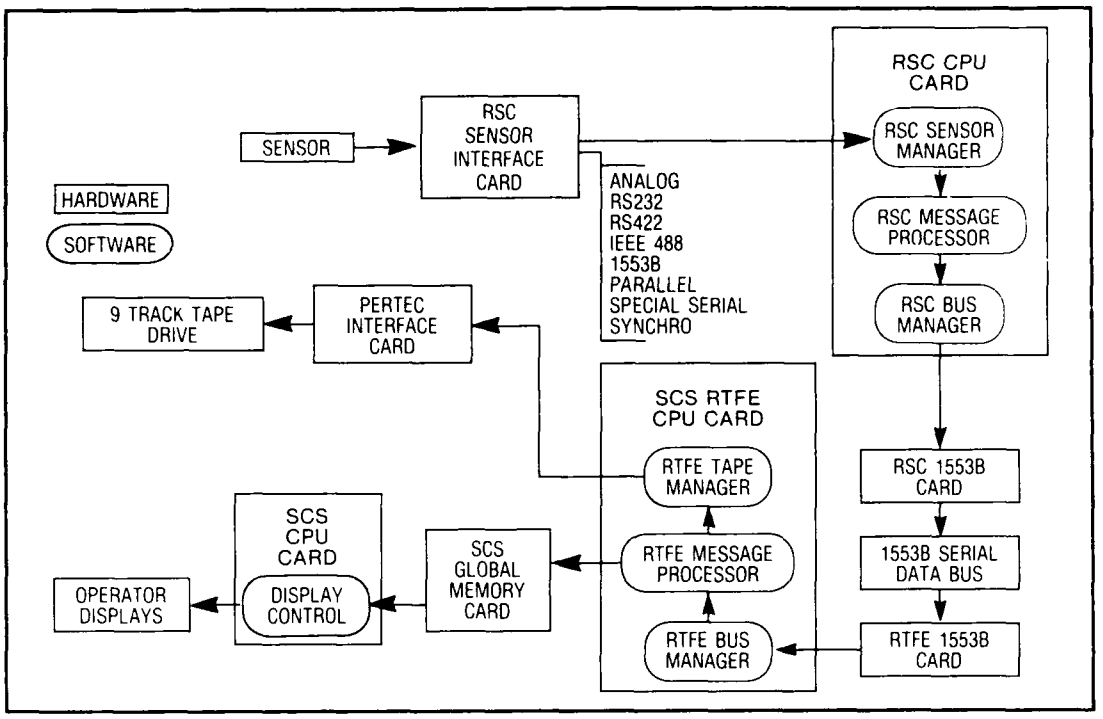

Figure 18. GASS data flowpath.

realize that the only widely accepted method for ensuring reliability of software is through exhaustive testing. However, some of the costs of software debugging can be reduced by following a rigorous software quality assurance plan.

A systematic approach to the design of GASS involved first the identification of the system's requirements and major design restrictions. Then, a methodical analysis of available architecture options was made to produce a design that meets both the requirements and restrictions. The major elements of the system's architecture are the number of processors, the way these processors are distributed and interconnected in the system, and the distribution of the software tasks. This study revealed that the design of embedded real-time distributed systems is largely application specific; the design of GASS given in this paper offers one possible solution to the given design parameters.

V. Recommendations

A. Further Study

Once GASS is completed and has undergone successful functional testing, performance testing should be done to evaluate the system's design and its potential for expansion. One key performance criterion is the loading, or the "busy time" of the system's processors. This measure will reveal critically loaded processors, if they exist, and will also provide an idea of how much extra load the system can handle. If there are critically loaded processors in the system, then the problem can probably be remedied by a different distribution of sensors in the RSCs. Another significant measure is the latency time of the network bus. Excessive bus latency can adversely affect the navigation function of the system, since GASS will be used on an aircraft. At typical survey speeds of 200 knots, 15 seconds is equivalent to 0.95 miles of travel. Probably the most significant measure of system performance at this point is that of measurement time skew. Even though the system is time synchronized and sensor acquisition tasks are initiated within a few microseconds of each other, a time skew still exists between the actual measurements. Part of this skew can be accounted for by the fact that the acquisition tasks for different sensors require varying lengths of time to actually trigger their sensors. The major contributor to the time skew, and also the hardest to measure, is the time required for each instrument to perform a measurement once triggered. Some instruments, the barometric altimeters, for example, provide data almost instantly after triggering of the acquisition task. Other instruments, such as the frequency counters used for the ASQ-81 magnetometer, may take almost the entire sample interval (when at an 8- or 16-Hz sampling rate) to make the measurement. These time skews need to be accurately determined so that they can be accounted for during post processing of the GASS data.

B. Future System Enhancements

The GASS architecture will allow very easy modification and modernization of the system. If processors become overloaded, then the system can be easily reconfigured to distribute the load properly. If load distribution is not an adequate solution, then more processors could be added to the existing nodes, or extra nodes can be added to the system. If bus latency becomes a problem, then bus performance can be improved by using a token passing MAP instead of the currently used controller scheme. In the event of severe bus loading problems, the system could be segmented into several buses by adding more bus interface cards to the SCS, or the entire bus system could be upgraded to a faster fiber-optic media. The system's fault detection capabilities can be improved easily by incorporating cross-instrument data testing, as discussed by Bourgeois[48]. System availability can be improved by altering the current RSC software structure. The current design allows any of the RSC processor cards to be used in any RSC. This design could be improved further by configuring the RSC software to allow execution of any combination of the system's sensor tasks. With this change, many of the sensors of a failed RSC could be relocated to an operational RSC.

VI. References

1. Harris, M., B. Bourgeois, P. Wischow, and J. Ross (1989). *Geophysical Airborne Survey System — System Overview*. In prep. Naval Oceanographic and Atmospheric Research Laboratory, Stennis Space Center, MS.

2. Langley, F., D. Siegel, W. Savage, and R. Wehman (1979). Federated microcomputer systems for on-board missile guidance and control. *AGARD Advances in Guidance and Control Systems using Digital Techniques*, May.

3. Wilcock, G., P. Lancaster, and C. Moxey (1982). Integrated control of mechanical systems for future combat aircraft. *AGARD, Tactical Airborne Distributed Computing and Networks.*

4. Shin, K. and C. Krishna (1982). Performance study of a distributed microprocessor architecture for use aboard a military aircraft. *AGARD, Tactical Airborne Distributed Computing and Networks.*

5. De Feo, P., L. Geiger, and J. Harris (1985). *Intelligent Redundant Actuation System Requirements and Preliminary System Design.* Contractor Report, NASA Contract No. NAS2-12081.

6. Shin, K., R. Throne, and Y. Muthuswamy (1987). Communication and Control in an Integrated Manufacturing System. *Proceedings of the 1st Annual Workshop on Space Operations Automation and Robotics*, pp. 405-411.

7. Gluch, D. and M. Paul (1986). Fault-Tolerance in Distributed Fly-by-Wire Flight Control Systems. *Proceedings of the AIAA/IEEE 7th Digital Avionics Systems Conference*, pp. 507-514, October.

8. Kieckhafer, R., C. Walter, A. Finn, P. Thambidurai (1988). The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37(4):398-405.

9. Fura, D., T. Hill, and M. Raftery Design and Validation of the IFTAS Fault-Tolerant Clock. *Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference*, pp. 235-242, October.

10. Svenningsson, M. (1987). Central processing unit for fault tolerant computing in Columbus. *Acta Astronautica*, 15:661-665.

11. Madden, W. and P. Wilhelm (1988) Space Station Data Management System Architecture. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 792-798, October.

12. Whitelaw, V. (1988). The Space Station Data Management System: Avionics That Integrate. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 767-774, October.

13. Mejzak, R. (1987). New technology impacts on future avionics architectures. *AGARD, Advanced Computer Aids in the Planning and Execution of Air Warfare and Ground Strike Operations.*

14. Pedar, A. and V. Sarma (1983). Architecture optimization of aerospace computing systems. *IEEE Transactions on Computers*, C-32:(10)911-922.

15. Vielcanet, P., H. Horgen, T. Demoy, and P. Howlwett (1984). *Comparative Study on Data System Architectures.* Contract Report, ESTEC Contract No. 5524/83/NL/PP.

16. Mangoubi, R., E. Gai, and B. Walker (1986). On the Performance Analysis of Real-Time Distributed Computer System. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 529-535, October.

17. Lala, J., L. Alger, S. Adams, L. Burkhardt, G. Nagle, and N. Murray (1988). Development of a Space-Systems Network Testbed. *Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference*, pp. 571-579, October.

18. Brock, L., and J. Deyst (1988). Modular Avionics Systems Studies. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 1-7, October.

19. Gelderloos H., and D. Wilson (1976). Redundancy Management of Shuttle Flight Control Sensors. *Proceedings of the 15th IEEE Conference on Decision and Control*, pp. 462-475, December.

20. Watson, J., W. Yousey, and J. Railey (1979). Redundancy management considerations for a control-configured fighter aircraft triplex digital fly-by-wire flight control system. *AGARD Advances in Guidance and Control Systems Using Digital Techniques.*

21. Sievers, M., G. Kravetz, B. Dussia, and J. Jackson (1982). *Military Standard Fault-Tolerant Microcomputer.* Contractor Report, Office of Naval Research Contract No. N00014-82-C-0126.

22. McGlone, M., R. Miller, W. Davies, and P. Adams (1983). Full authority fault tolerant electronic engine control systems for advanced high performance engines. *Society of Automotive Engineers, Aerospace Congress and Exposition,* October.

23. Hartman, G., J. Wall, E. Rang, H. Lee, R. Schulte, and W. Ng (1983). *Advanced Flight Control System Study.* Contract Report, Ames Research Center, Contract NAS4-2876.

24. Dzwonczy, M. and H. Stone (1988). A Fault-Tolerant Suite for an Entry Research Vehicle. *Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference*, pp. 593-598, October.

25. Goldberg, J., et al. (1984). *Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer.* Contract Report, NASA Contract NASA-CR-172146.

26. Montgomery, V. (1981). *Development of a Fault-Tolerant Microprocessor Based Computer System for Space Flight.* Southern University, Baton Rouge, LA, Contract NSG-8053.

27. Evans, R. and S. Price (1981). Fault Tolerant Microprocessor System Design. *Proceedings of the 11th Annual International IEEE Symposium on Fault-Tolerant Computing*, pg 214-220, June.

28. Yaacob, M., M. Hartley, and P. Depledge (1983). Operational Fault-Tolerant Microcomputer for Very High Reliability. *IEE Proceedings*, 130.E(3):90-94.

29. Smith, T. (1984). Fault Tolerant Processor Concepts and Operation. *Proceedings of the 14th IEEE Fault Tolerant Computing Symposium*, pp. 158-163, June.

30. Ichikawa, S., Y. Kawada, M. Mine, Y. Ishige, and A. Itsukaichi (1988). Fault-Tolerant Computing System and Software for Engineering Test Satellite-VI (ETS-VI) Attitude and Control Subsystem. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 170-176, October.

31. Lamport, L., R. Shostak and M. Pease (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382-401.

32. Shepherd, J. (1982). A Method of Designing Fault Tolerant Software. *Proceedings of the Royal Aeronautical Society's Symposium on Certification of Avionic Systems*, April.

33. Hitt, E. (1986). Real-Time Fault Tolerant Software in Distributed Avionics Systems Architectures Using Digital Data Buses. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 523-528, October.

34. Avizienis, A. and M. Lyu (1988). On the Effectiveness of Multiversion Software in Digital Avionics. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 422-427, October.

35. Anderson, R. (1980). *Automated Recovery in a Real-Time, Distributed, Multiple Microprocessor Computer System.* Masters Thesis, Naval Postgraduate School, December.

36. Schneider, F. and R. Schlichting (1981). Towards Fault Tolerant Process Control Software. *Proceedings of the 11th Annual International IEEE Symposium on Fault-Tolerant Computing*, pg 48-55, June.

37. Clarke, E. and C. Nikolaou (1981). Reconfiguration Strategies for Reliable Shared Memory Multiprocessor Systems. *Proceedings of the 11th Annual International IEEE Symposium on Fault-Tolerant Computing*, pg 152-158, June.

38. Schmid, H., S. Larimer, and T. Madak (1986). Channelized or Nonchannelized Fault-Tolerant Computers: A Hardware Complexity Comparison of Fault-Tolerant Computers for Flight Control Systems. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 655-663, October.

39. Loques, O. and J. Kramer (1986). Flexible Fault Tolerance for Distributed Computer Systems. *IEE Proceedings*,133.E(6):319-322.

40. Best, D., K. McGahee, and R. Shultz (1988). A Fault Tolerant Avionics Multiprocessing System Architecture Supporting Concurrent Execution of Ada Tasks. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 256-262, October.

41. Eckhardt, D. (1985). Fault-Tolerant Software Experiment Objectives and Status. *NASA Computer Science/Data Systems Technical Symposium*, April.

42. Voigt, R. (1986). *The Design of a Real Time Operating System for a Fault Tolerant Microcomputer.* Masters Thesis, Navy Postgraduate School, December.

43. Dekker, G. (1985). *Reliability Aspects of Software for Digital Avionics.* Contractor Report for the Netherlands Dept. of Civil Aviation.

44. Cramer, M. (1986). A Quantitative Analysis of the History of Developing a Large Embedded Software System. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 355-362, October.

45. Cooper, L., J. Radatz, and R. Butler (1986). DoD Develops Single Set of Software Quality Standards. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 777-783, October.

46. Smith, J. (1986). Software Quality Issues. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 765-768, October.

47. Anonymous (1983). *Software Quality Assurance for ESA Spacecraft and Associated Equipment.* European Space Research and Technology Centre Report, February, ASA Document #84N26037.

48. Bourgeois, B., M. Harris, T. Nicolaides, P. Wischow, A. Martinez, P. Duvoisin, and R. Drake (1989). Fault Detection and Reliability of the GASS Multi-processor Survey System. *Proceedings of IEEE Southeast Conference*, pp. 319-325, April.

49. IEEE Standard Digital Interface for Programmable Instrumentation (1987). ANSI/IEEE Standard 488.1.

50. Fritz, J., C. Kaldenbach, and L. Progar (1985). *Local Area Networks Selection Guidelines.* Information Systems & Networks Corporation, Prentice Hall, Englewood Cliffs, New Jersey.

51. Shaw, J., H. Herzog, and K. Okubo (1986). Digital Autonomous Terminal Access Communication (DATAC). *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 221-226, October.

52. Holmes, D. (1986). Global System Data Bus Using the Digital Autonomous Terminal Access Communication Protocol. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 227-233, October.

53. McGough, J. (1986). Evaluation of Data Busses for Flight Critical Control Applications. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 718-727, October.

54. Anderson, S. (1986). The High Speed Interconnect System Architecture and Operation. *Proceedings of the Aerospace Avionics Equipment and Integration Conference*, pp. 173-183, April.

55. Spieth, J. and W. Seward (1986). Simulation Model of a High-Speed Token-Passing Bus for Avionics Applications. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 242-249, October.

56. Meyer, J. (1986). SAE AE-9B Draft Standard High Speed Token Passing Data Bus for Avionics Applications. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 234-241, October.

57. Nelson, J., L. Shafer, D. Hamlin, and H. Herrmann (1987). A Candidate for Linear Token-Passing, High-Speed Data Bus Systems. *Proceedings of the 2nd Aerospace Avionics Equipment and Integration Conference*, pp. 105-120, November.

58. Ludvigson, M., M. Modrow, and P. Goldman (1988). The High Speed Bus Technology Development Program. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, October.

59. Kroeger, B. and H. Shih (1988). An SAE High Speed Ring Bus Overview. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 719-723, October.

60. Werner, B. (1989). Token-Ring Local-Area Networks and their Performance. *Proceedings of the IEEE*, 77(2):238-256.

61. Cooling, J. (1986). *Real-Time Interfacing: Engineering Aspects of Microprocessor Peripheral Systems.* Van Nostrand Reinhold Co. Ltd., Berkshire, England.

62. Keen, W. (1988). Ada in Avionics — Beyond Validation. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 251-255, October.

63. Lala, J., L. Alger, R. Gauthier, and M. Dzwonczyk (1986). A Fault Tolerant Processor to Meet Rigorous Failure Requirements. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 555-562, October.

64. Voelcker, J. (1987). Ada: from promise to practice? *IEEE Spectrum*, pp. 44-49, April.

65. Jennings, R. (1986). Avionics Standard Communication Bus, It's Implementation and Usage. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 242-249, October.

66. Blaha, M., C. DeGennaro, and S. Utley (1988). A Dual Speed, Mil-Std-1553B Compatible Fiber Optic Data Bus. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 395-398, October.

67. Humphrey, T. (1988). Reducing the Risks of Using Ada Onboard the Space Station. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 599-602, October.

68. Lahn, T., S. Minear, and J. Murray 1986). Some Views on the Use of Ada for Digital Flight Control Systems. *Proceedings of the 7th AIAA/IEEE Digital Avionics Systems Conference*, pp. 455-460, October.

69. Garlington, K. and A. Tyrrell (1988). A Case Study: F-16 ADA Digital Flight Control System. *Proceedings of the AIAA/IEEE 8th Digital Avionics Systems Conference*, pp. 267-272, October.

70. Harris, M., H. Mesick, H. Byrnes, T. Curran, and V. Contarino (1987). A Laser Sounder for U.S. Navy Hydrographers. *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium.*

Distribution List

Asst Secretary of the Navy
(Research, Engineering & Systems)
Navy Department
Washington DC 20350-1000

Chief of Naval Operations
Navy Department
Washington DC 20350-1000
Attn: OP-02
OP-71
OP-0962X
OP-987

Oceanographer of the Navy
Chief of Naval Operations
U.S. Naval Observatory
34th & Mass Ave., NW
Washington DC 20390-1800
Attn: OP-96

Commander
Naval Air Development Center
Warminster PA 18974-5000

Commanding Officer
Naval Coastal Systems Center
Panama City FL 32407-5000

Commander
Space & Naval Warfare Sys Com
Washington DC 20363-5100

Commander
Naval Facilities Eng Command
Naval Facilities Eng Command HQ
200 Stovall St.
Alexandria VA 22332-2300

Commanding Officer
NOARL
Stennis Space Center MS 39529-5004
Attn: Code 100
Code 105
Code 115
Code 117, J. Hammack
Code 125EX
Code 125L (10)
Code 125P
Code 200
Code 300
Code 350, D. Hickman
Code 351, J. Byrnes
Code 351, J. Braud
Code 351, M. Lohrenz
Code 351, B. Bourgeois
Code 351, M. Harris
Code 351, P. Wischow
Code 351, J. Ross
Code 400

Brooke Farquhar
NOARL Liaison Office
Crystal Plaza #5, Room 802
2211 Jefferson Davis Hwy.
Arlington VA 22202-5000

Commanding Officer
Naval Research Laboratory
Washington DC 20375

Commander
Naval Oceanography Command
Stennis Space Center MS 39529-5000

Commanding Officer
Fleet Numerical Oceanography Center
Monterey CA 93943-5005

Commanding Officer
Naval Oceanographic Office
Stennis Space Center MS 39522-5001
Attn: Code GGAP, B. Mullen
Code A, J. Depner

Commander
Naval Ocean Systems Center
San Diego CA 92152-5000

Commanding Officer
ONR Branch Office
Box 39
FPO New York NY 09510-0700

Commander
David W. Taylor Naval Research Center
Bethesda MD 20084-5000

Commander
Naval Surface Weapons Center
Dahlgren VA 22448-5000

Commanding Officer
Naval Underwater Systems Center
Newport RI 02841-5047

Superintendent
Naval Postgraduate School
Monterey CA 93943

Director of Navy Laboratories
Rm 1062, Crystal Plaza Bldg 5
Department of the Navy
Washington DC 20360

Officer in Charge
New London Laboratory
Naval Underwater Sys Cen Det
New London CT 06320

Director
National Ocean Data Center
WSC1 Room 103
6001 Executive Blvd.
Rockville MD 20852
Attn: G. W. Withee

Director
Woods Hole Oceanographic Inst
P.O. Box 32
Woods Hole MA 02543

University of California
Scripps Institute of Oceanography
P.O. Box 6049
San Diego CA 92106

Officer in Charge
Naval Surface Weapons Center Det
White Oak Laboratory
10901 New Hampshire Ave.
Silver Spring MD 20903-5000
Attn: Library

Commanding Officer
Fleet Anti-Sub Warfare Training Center,
Atlantic
Naval Station
Norfolk VA 23511-6495

Defense Mapping Agency Sys Cen
12100 Sunset Hill Rd. #200
Reston VA 22090-3207
Attn: SGWN
M. Wagner
E. Danford

Office of Naval Technology
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 20, Dr. P. Selwyn
Code 228, Dr. M. Briscoe
Code 234, Dr. C. V. Votaw

Office of Naval Research
800 N. Quincy St.
Arlington VA 22217-5000
Attn: Code 10
Code 10D/10P, Dr. E. Silva
Code 12
Code 112, Dr. E. Hartwig

Commander
Naval Sea Systems Command
Naval Sea Systems Command HQ
Washington DC 20362-5101

Commanding Officer
Naval Civil Engineering Laboratory
Port Hueneme CA 93043

Commander
Naval Air Systems Command
Naval Air Systems Command HQ
Washington DC 20361-0001

Pennsylvania State University
Applied Research Laboratory
P.O. Box 30
State College PA 16801

University of Texas at Austin
Applied Research Laboratories
P.O. Box 8029
Austin TX 78713-8029

Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Rd.
Laurel MD 20707

University of Washington
Applied Physics Laboratory
1013 Northeast 40th St.
Seattle WA 98105

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. Agency Use Only *(Leave blank).*	2. Report Date.	3. Report Type and Dates Covered.
	April 1990	Final

4. Title and Subtitle.

Design of a Distributed Microprocessor Sensor System

5. Funding Numbers.

Program Element No. 980101

Project No.

Task No.

Accession No. 93519G

6. Author(s).

Brian S. Bourgeois, Mike M. Harris, Perry B. Wischow, and Jerry H. Ross

7. Performing Organization Name(s) and Address(es).

Ocean Science Directorate
Naval Oceanographic and Atmospheric Research Laboratory
Stennis Space Center, Mississippi 39529-5004

8. Performing Organization Report Number.

NOARL Report 1

9. Sponsoring/Monitoring Agency Name(s) and Address(es).

10. Sponsoring/Monitoring Agency Report Number.

11. Supplementary Notes.

12a. Distribution/Availability Statement.

Approved for public release; distribution is unlimited. Naval Oceanographic and Atmospheric Research Laboratory, Stennis Space Center, Mississippi 39529-5004.

12b. Distribution Code.

13. Abstract *(Maximum 200 words).*

The Geophysical Airborne Survey System (GASS), developed by the Naval Ocean Research and Development Activity*, is a real-time, distributed microprocessor sensor system. The Naval Oceanographic Office intends to use this system on the Project Magnet P-3 Orion aircraft to collect worldwide magnetic and hydrographic data.

This report briefly discusses the mission and the history of GASS and reviews the technology advances of the last decade in the area of real-time distributed microprocessor systems. Specific topics include the goals of distributed system design, the qualitative value of distributed system design, and reliable design techniques for the hardware and software in distributed systems. This technology review provides a foundation for the GASS design.

The system design decisions made for GASS are detailed, and the system's architecture is described. Detailed drawings and descriptions of the hardware and software for the final GASS design are also included. Recommendations are made for possible system enhancements and for areas that require further investigation.

*Now the Naval Oceanographic and Atmospheric Research Laboratory (NOARL).

14. Subject Terms.	15. Number of Pages.
distributed microprocessor systems, GASS, systematic design, system architecture	24
	16. Price Code.

17. Security Classification of Report.	18. Security Classification of This Page.	19. Security Classification of Abstract.	20. Limitation of Abstract.
Unclassified	Unclassified	Unclassified	None